**中央财经大学**

**“数据结构”实验报告**

**实验项目名称**  栈和队列的操作及应用

**所属课程名称**  数据结构与算法

**实 验 日 期**  2023-3-26

**姓 名**  陈博驹

**班 级**  金融工程21-2

**学 号**  2021310277

**成 绩**

|  |
| --- |
| **实验概述：** |
| **【实验目的及要求】**   1. 熟悉Visual Studio 开发及调试环境。 2. 掌握队列和栈的基本操作，体会栈和队列的特点。 3. 练习栈和队列的简单应用，体会数据结构对算法复杂度的影响。   **【实验环境】**  写出程序编写时所使用的编译器、操作系统等。  **VSC**  **Win10** |
| **实验内容：** |
| **【实验内容与实验过程及分析】**  列出实验需要做的各个题目，并针对每个题目：  1. 写出关键代码（要求选出该题目最核心的代码并加注释）；  2. 描述调试的过程、调试过程中所出现的现象及解决方法，给出题目最终运行结果；  3. 对该题目进行简短总结，写出该题目所涉及的知识点。   1. **将顺序栈（数据元素为char类型）的定义补充完整。**   [关键代码]  typedef struct /\* 定义字符顺序栈 \*/  {  SElemType data[MAXSIZE];  int top;  }SqStack;  typedef struct//定义数字顺序栈  {  int data[MAXSIZE];  int top;  }StackNum;  [实验运行结果]    [实验总结(题目所涉及的知识点)]   * **对栈基本定义的了解，包括后进先出的理解等，栈的定义应该包含两部分内容：包括栈顶指针，但是实际上并不是指针变量，实际上是一个整型变量，由于我们实际上是使用数组来表示栈的结构，实际上栈顶指针这个整型变量起到的作用是在数组表示最后那个元素即栈顶元素的位置；另一部分内容及一个数组，里面包含着栈所存储的元素类型（在此处时字符型变量）。**   **2. 实现顺序栈的Push和Pop操作。**  [实验代码]  int Push(SqStack\* S, SElemType e) /\* 插入函数将SElemType插入符号栈S \*/  {  if (S->top == MAXSIZE - 1)  return ERROR;  if (S->top != MAXSIZE - 1) {  S->top++;  S->data[S->top] = e;  return OK;  }    }  nt Pop(SqStack\*S, SElemType &e) /\* 弹出函数，将符号栈栈顶元素弹出并赋值给e \*/  {  if (S->top == -1)  return ERROR;  e= S->data[S->top];  S->top--;  return OK;  }  [实验运行结果]      [实验总结(题目所涉及的知识点)]   * 由于在参数设定上采取了顺序栈的指针变量，所以在函数体中要正确使用指针变量的形式。本质上来说，入栈操作先判断栈是否已满，若栈已满即栈顶指针的位置指向了MAXSIZE-1，程序报错，若未满，栈顶指针即整型变量+1，同时把插入元素e的值赋给栈顶元素的数列中。出栈操作先判断栈是否为空，若栈为空即栈顶指针指向了-1，程序报错，若未空，先把栈顶指针指向的数组位置的数值赋给e，再栈顶指针-1.   **3.** **算术表达式的求解（只包含+,-,\*,/）**  [实验代码]  void trans(char\* exp, char postexp[])//得到逆波兰式  {  char e; //给Pop与Gettop函数使用  SqStack\* Optr; //定义符号运算栈指针  int i = 0; // i作为postexp下表  InitSqStack(Optr); //初始化运算符栈  while (\*exp != '\0') //exp表示为扫描时循环  {  switch (\*exp)  {  case '(': //判定左括号  Push(Optr, '(');  exp++; //exp指针前移，继续处理下一个字符  break;  case ')': //判定为右括号  Pop(Optr, e);  while (e != '(') //直到遇到“（”为止  {  postexp[i++] = e;  Pop(Optr, e);  }  exp++;  break;  case '+': // 判定为“+”或“-”  case '-':  while (!StackEmpty(Optr)) //直到栈空或者栈顶为“（”，然后将其入栈  {  GetTop(Optr, e);  if (e == '(')  break;  else  {  postexp[i++] = e;  Pop(Optr, e);  }  }  Push(Optr, \*exp); //最后将“+”或者“-”入栈  exp++;  break;  case '\*': //判定为“\*”或“/”  case '/':  while (!StackEmpty(Optr)) //直到栈顶为空或者为“（”“\*”“/”，然后将其入栈·  {  GetTop(Optr, e);  if (e == '/' || e == '\*')  {  postexp[i++] = e;  Pop(Optr, e);  }  else  break;  }  Push(Optr, \*exp); //最后将“\*”“/”入栈  exp++;  break;  default: //处理数字字符  while (\*exp >= '0' && \*exp <= '9') //循环处理多位数字  {  postexp[i++] = \*exp;  exp++;  }  postexp[i++] = ' '; //以空格 表示一个数字串结束  }  }  while (!StackEmpty(Optr)) //结束exp扫描，栈不空时循环  { //将optr的所有运算符依次出栈保存到postexp  Pop(Optr, e);  postexp[i++] = e;  }  postexp[i] = '\0'; //给postexp添加结束标志  DestroyStack(Optr); //销毁栈  }  //计算后缀表达式的值  int compvalue(char\* postexp)  {  StackNum\* Opnd; //定义数字栈  InitStackNum(Opnd); //初始化数字栈  int result; //存放结果  int a, b; //弹出栈的两个数  int c; //计算弹出两个数的运算结果  int d; //将连续的数字字符转化为十进制整数保存在d中  while (\*postexp != '\0') //postexp未扫描完时循环  {  switch (\*postexp)  {  case '+': //判定为加号  pop(Opnd, a); //从Open中弹出两个数值a，b，c=a+b，将c压入栈  c = b + a;  push(Opnd, c);  break;  case '-': //判定为减号  pop(Opnd, a); //从Open中弹出两个数值a，b，c=b-a,将c压入栈  pop(Opnd, b);  c = b - a; //b-a因为b后出栈  push(Opnd, c);  break;  case '\*': //判定为乘号  pop(Opnd, a); //从Opnd弹出两个数值a和b，c=a\*b，将c入栈  c = b \* a;  push(Opnd, c);  break;  case '/': //判定为除号  pop(Opnd, a); //从Opnd中弹出两个数值a和b，若a不等于0，则c=b/a，将c入栈  pop(Opnd, b);  if (a != 0) //检查除数的合法性  {  c = b / a;  push(Opnd, c);  break;  }  else  {  cout << "\n除零错误！" << endl;  exit(0);  }  default: //处理数字字符  d = 0; //将连续的数字字符转换为对应数值存放到d中  while (\*postexp >= '0' && \*postexp <= '9')  {  d = 10 \* d + \*postexp - '0';  postexp++;  }  push(Opnd, d);  break;  }  postexp++; //继续处理下一个字符  }  pop(Opnd, result); //弹出运算最终的结果  delete Opnd; //销毁栈  return result; //返回结果  }  [实验运行结果]    [实验总结(题目所涉及的知识点)]  第一个函数 **trans** 的功能是将中缀表达式转换为后缀表达式，参数 **exp** 是一个字符串类型的中缀表达式，参数 **postexp** 是一个字符数组类型的后缀表达式。该函数的大致流程如下：   1. 定义一个字符类型的变量 **e** 用于后面的出栈操作，一个指向运算符栈的指针 **Optr**，以及一个整型变量 **i** 用于遍历后缀表达式数组。 2. 初始化运算符栈。 3. 使用 while 循环扫描中缀表达式中的每个字符，直到遇到字符串结尾标志 **\0**。 4. 对于每个字符，使用 switch 语句进行判断，分别处理左右括号、加减乘除和数字字符。具体来说：    * 如果是左括号，将其压入运算符栈，并将 **exp** 指针指向下一个字符。    * 如果是右括号，弹出运算符栈中的元素直到遇到左括号，将弹出的运算符依次加入后缀表达式中，然后将 **exp** 指针指向下一个字符。    * 如果是加减号，弹出运算符栈中的所有优先级不低于加减号的元素，加入后缀表达式中，最后将加减号入栈。然后将 **exp** 指针指向下一个字符。    * 如果是乘除号，弹出运算符栈中的所有优先级不低于乘除号的元素，加入后缀表达式中，最后将乘除号入栈。然后将 **exp** 指针指向下一个字符。    * 如果是数字字符，将其加入后缀表达式中，直到遇到一个非数字字符，然后在数字字符末尾添加一个空格，表示一个数字串结束。 5. 使用 while 循环弹出运算符栈中的所有元素，加入后缀表达式中。 6. 最后在后缀表达式的末尾添加一个字符串结尾标志 **\0**，然后销毁运算符栈。   第二个函数 compvalue 的功能是计算后缀表达式的值，参数 postexp 是一个字符数组类型的后缀表达式。该函数的大致流程如下：  定义了一个函数 compvalue，它的参数是一个指向字符数组的指针，表示后缀表达式。声明了一些变量，包括一个 StackNum 类型的指针 Opnd，表示数字栈；一个整数 result，用于存储最终的结果；两个整数 a 和 b，用于弹出数字栈的两个数；一个整数 c，用于计算弹出两个数的运算结果；一个整数 d，用于将连续的数字字符转化为十进制整数保存在其中。当postexp指向的字符既不是加号、减号、乘号、除号时，说明该字符为数字字符，需要将连续的数字字符转换为对应的数值，存放到变量d中。这里用了一个while循环，只要postexp指向的字符是数字字符，就将其转换为对应的数值并存放到d中。具体的实现方法是将d的值先乘以10，然后加上当前字符对应的数字，最后将postexp指向下一个字符，继续处理下一个数字字符，直到遇到不是数字字符为止。如果字符不是数字字符也不是加减乘除符号，就直接跳到下一个字符，继续处理。最后，弹出数字栈中的最后一个元素作为结果返回，并销毁数字栈。  **4.** **顺序循环队列的定义和基本操作**  [实验代码]  typedef struct  {  QElemType data[MAXSIZE];/\*一维数组表示队列，容量为 MAXSIZE \*/  int front; /\*表示队头元素位置\*/  int rear; /\*表示队尾元素位置\*/  } SqQueue;  Status InitQueue(SqQueue\*Q)/\*队列的初始化\*/  {  Q->front=0;  Q->rear=0;  return OK;  }  int LengthOfQueue(SqQueue Q)/\*求队列的长度\*/  {  return (Q.rear-Q.front+MAXSIZE)%MAXSIZE;  }  Status InsertQueue(SqQueue \*Q,QElemType e)  {  if((Q->rear+1)+1%MAXSIZE==Q->front)  {  return ERROR;//此时队满  }  Q->data[Q->rear]=e;  Q->rear=(Q->rear)+1%MAXSIZE;/\*rear指针后后移，若到最后则转到数组头部\*/  return OK;  }  Status DeleteQueue(SqQueue \*Q,QElemType e)/\*元素出列并且将出列元素的值赋给e\*/  {  if(Q->front==Q->rear)  {  return ERROR;  }  e=Q->data[Q->front];  Q->front=(Q->front+1)%MAXSIZE;  return OK;  }  int IsEmpty(SqQueue \*Q)/\*判断队列是否为空\*/  {  if(Q->front==Q->rear)  {  return OK;  }  else  {  return ERROR;  }  }  int GetHead(SqQueue \*Q,QElemType e)/\*取队头元素\*/  {  if (Q->front==Q->rear)  {  return ERROR;  }  e=Q->data[Q->front];  return OK;  }  int DesttroyQueue(SqQueue\*Q)/\*销毁队列\*/  {  if(Q->data)  {  Q->front=0;  Q->rear=0;  delete(Q->data);  }  return OK;  }  [实验运行结果]    [实验总结(题目所涉及的知识点)]  顺序循环队列的重要注意事项：要留一个空的数组格子来区分队空和队满。  队列的定义包含3个成员，一个是用来装数据的数组，一个整型变量表示队头元素位置。一个整型变量表示队尾元素位置。之后队列的初始化即构建一个空表，使得front和rear都为0.求队列长度的公式为：(Q.rear-Q.front+MAXSIZE)%MAXSIZE;入队时，要先判断队列是否已满，公式为(Q->rear+1)+1%MAXSIZE==Q->front，若已满程序报错；若不满，将e赋值给队尾整型变量rear指向的数组数据，之后rear向后移动一位，注意循环需要模队列长度。元素的出列需要判断队列是否为空，公式为：Q->front==Q->rear，若为空程序报错；若不为空，将此时队头整型变量指向的数组元素赋值给e，并且将队头整型变量指针向后移动一位注意循环要取模队列长度。  **5. （选做）约瑟夫环问题（假设按最初位置从1开始顺序编号，n和m由用户给出）**  [实验代码]  #include<iostream>  using namespace std;  struct cQueue/\*队列结构体的定义\*/  {  int \*a=NULL;  int maxSize;  int front,rear;    };  void init(cQueue\* q, int sz)/\*初始化一个队列\*/  {  q->a=new int[sz];//创建一个新的整型数组  q->maxSize=sz;//队列的最大长度为队列的长度  q->front=0;  q->rear=0;  }  void flush(cQueue\* q)/\*删除队列\*/  {  delete q->a;  }  bool empty(cQueue\* q)/\*判断队列是否为空\*/  {  return q->front==q->rear;  }  bool full(cQueue\* q)/\*判断队列是否为满\*/  {  return q->front==(q->rear+1)%q->maxSize;  }  void clear(cQueue\* q)/\*将队列清空\*/  {  q->front=0;  q->rear=q->front;  }  int size(cQueue\* q)/\*计算队列的长度\*/  {  return (q->rear-q->front+q->maxSize)%q->maxSize;  }  bool push(cQueue\* q, int x)/\*将数据x入列\*/  {  if(full(q))return false;  q->a[q->rear]=x;  q->rear=(q->rear+1)%q->maxSize;  return true;  }  void pop(cQueue\* q)/\*将数据出列\*/  {  q->front=(q->front+1)%q->maxSize;  }  int front(cQueue\* q)/\*得到开头的数据\*/  {  return q->a[q->front];  }  int main()  {  cQueue q;  cQueue ou;    cQueue \*q1=&q;  cQueue \*out=&ou;  cout<<"请分别输入总人数，开始的号数和每几个人淘汰一次"<<endl;  int n,m,w;//n:num of people;m:start people;w:outpost num;  cin>>n>>m>>w;  init(q1,n+5);  init(out,n+5);  for(int i=1;i<=n;i++)  {  push(q1,i);  }  int i=0;  int w1=w;  while(w1!=1){  push(q1,front(q1));  pop(q1);  w1--;  }    while(size(q1)!=1)  {  i++;  if(i%w!=0){  push(q1,front(q1));  pop(q1);  }  else{  push(out,front(q1));  pop(q1);  }    }  int n0=size(out);  cout<<"出列顺序为：";  for(int j=0;j<n0;j++)  {  cout<<front(out)<<' ';  pop(out);  }  cout<<endl;  cout<<"最后剩下的"<<front(q1)<<"号\n";  flush(q1);  flush(out);  system("pause");  return 0;  }  [实验运行结果]    [实验总结(题目所涉及的知识点)]  这段代码实现了根据约瑟夫问题的规则，将队列中的元素出队并重新入队，直到队列中只剩下最后一个元素。  具体的实现过程如下：  1.定义变量 i 和 w1，其中 i 表示当前的出队次数，w1 表示每轮出队的人数。  2.当 w1 不为 1 时，将队首元素出队并重新入队 w1 - 1 次，实现每轮出队的人数减少一个的功能。  3.当队列中元素个数大于 1 时，进行出队并重新入队操作，具体步骤如下：  a. 对于第 i + 1 个人（i 从 0 开始计数），如果 i + 1 不是每 w 个人出队的位置，则将队首元素出队并重新入队。  b. 对于第 i + 1 个人，如果 i + 1 是每 w 个人出队的位置，则将队首元素出队并加入另一个队列 out 中，实现每 w 个人出队的功能。  4.循环结束后，输出出队的顺序和最后剩下的人的编号。  总体来说，这段代码实现了约瑟夫问题的基本思路，即按照一定的规则不断出队和重新入队，直到最后只剩下一个人。其中利用队列的数据结构特点，方便地实现了每轮出队和每 w 个人出队的功能。  **6. 链栈的基本操作：定义链栈结构并实现链栈的创建、销毁、push和pop操作**  [实验代码]  #include <iostream>  using namespace std;  typedef int SElemType;  typedef int Status; /\* Status 是函数类型，其值位函数执行结果的状态代码，如ERROR、OK、TRUE、FALSE等 \*/  # define MAXSIZE 20  # define ERROR 0  # define OK 1  # define TRUE 1  # define FALSE 0  # define NUll 0  typedef struct SNode  {  SElemType data;  struct SNode\* next;  }SNode;  typedef struct LinkStack  {  SNode\* top;  int length;  }LinkStack;  Status Push(LinkStack& S, SElemType e)//插入元素  {  SNode\* p = new SNode;  p->data = e;  p->next = S.top;  S.top = p;  S.length++;  return OK;  }  Status Pop(LinkStack& S, SElemType e)//压出元素  {  SNode\* p = new SNode;  p = S.top;  e = p->data;  S.top = p->next;  delete(p);  return OK;  }  Status Destroy(LinkStack& S)//销毁链栈  {  S.length = 0;  return OK;  }  Status LinkStackCrt(LinkStack &S) //创建链栈  {  SNode\* head = new SNode;  S.length = 0;  int n;  cout << "请输入链栈长度：";  cin >> n;  SElemType a;  head->next = NULL;  for (int i = 1; i <= n; i++) {  SNode\* p = new SNode;  cin >> a;  p->data = a;  p -> next = head->next;  S.top = p;  head->next = S.top;  S.length++;  }  return OK;  }  void LinkStackPrint(LinkStack &S) /\*输出带头结点的链栈\*/  {  SNode\* p=new SNode;  p->next=S.top;  cout << "\n目前该链表长度为" << S.length << endl;  int i;  i = 0;  cout << "\n链表的元素依次为：";  while (p->next != NULL)  {  ++i;  p = p->next;  cout << " 结点" << i << "=" << p->data;  }  cout << endl;  }  int main()  {  LinkStack S;  if (!LinkStackCrt(S))  cout << endl << "链栈未成功生成！" << endl;  else  {  LinkStackPrint(S);  char cmd;  SElemType x = 0;  do  {  cout << endl << "请选择操作：" << endl;  cout << "i , i ----- 压入元素" << endl;  cout << "d , d ----- 弹出元素" << endl;  cout << "q , q ----- 退出程序" << endl;  do  {  cmd = getchar();  } while ((cmd != 'i') && (cmd != 'd') && (cmd != 'q'));  switch (cmd)  {  case 'i':  cout << "\n请输入压栈数据（整型）: ";  cin >> x;  if (Push(S, x))  cout << "\n已压入元素" << x << endl;  else  cout << "\n压入元素失败！" << endl;  LinkStackPrint(S);  break;  case 'd':  Pop(S, x);  cout << "\n已弹出栈顶元素" << x << endl;  LinkStackPrint(S);  break;  }  } while (cmd != 'q');  }  system("pause");  return 0;  }  [实验运行结果]      [实验总结(题目所涉及的知识点)]  链栈的定义需要先定义链栈的每个结点：每个结点有两个成员，一个是数据域，另一个是指向下一个结点的指针，之后再定义链栈，有两个成员一个是链栈的长度另外一个是链栈的栈顶指针。插入元素操作必须先new一个新的结点指针p出来，将e的数值赋值给p的数据域，之后将p的下一个结点指向此时链栈的栈顶结点S.top，之后p结点称为新的S.top，最后S的length长度+1.弹出元素的操作是先new一个新的结点指针p来临时存储，p此时被赋值为S.top，同时将p的数据域的值赋值给e，新的S.top是p指向的下一个结点，最后释放p结点。销毁链栈即令链栈的长度为0.创建链栈必须先new一个头结点出来，之后令链栈的长度为0，通过键盘读入链栈的长度n，之后令头结点head指向NULL，再进行一个for循环，当i=1，且小于等于n时，每次i自增1，for循环中，new一个新结点p，p的数据域是从键盘读入的整型变量a，p指向的下一个结点时原来head结点指向的下一个结点，之后p成为了链栈的栈顶结点，同时新的head结点指向了栈顶结点p，链栈的长度+1. |
| **附：编译器错误信息** |
|  |